NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TWIDDLENET: METADATA TAGGING AND DATA DISSEMINATION IN MOBILE DEVICE NETWORKS

by

Christopher T. Clotfelter
Jonathon E. Towle

September 2007

Thesis Advisor:	Gurminder Singh
Co-Advisor:	Arijit Das

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2007	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
Twiddlenet: Metadata Tagging and Data Dissemination in Mobile Device Networks

5. FUNDING NUMBERS

6. AUTHOR(S)
Clotfelter, Christopher T.
Towle, Jonathon E.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Current mobile devices are much more than the limited modality communication tools or digital assistants they were only a few years ago; instead they offer a range of content capture capabilities, including high resolution photos, videos and sound recordings. Their communication modalities and processing power have also evolved significantly. Modern mobile devices are very capable platforms, many surpassing their desktop cousins only a few years removed. TwiddleNet is a distributed architecture of personal servers that harnesses the power of these mobile devices, enabling real time information dissemination and file sharing of multiple data types from commercial-off-the-shelf platforms.

This thesis focuses on two specific issues of the TwiddleNet design; metadata tagging and data dissemination. Through a combination of automatically generated and user input metadata tag values, TwiddleNet users can locate files across participating devices. Metaphor appropriate custom tags can be added as needed to insure efficient, rich and successful file searches. Intelligent data dissemination algorithms provide context sensitive governance to the file transfer scheme. Smart dissemination reconciles device and operational states with the amount of requested data and content to send, enabling providers to meet their most pressing needs, whether that is continuing to generate content or servicing requests.

14. SUBJECT TERMS Mobile file sharing, Distributed computing, Peer-to-peer networking, Tagging, Metadata, Data dissemination	15. NUMBER OF PAGES 87
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

TWIDDLENET: METADATA TAGGING AND DATA DISSEMINATION IN MOBILE DEVICE NETWORKS

Jonathan E. Towle
Captain, United States Maine Corps
B.S., California State University Humboldt

Christopher T. Clotfelter
Lieutenant, United States Navy
B.S., University of Tennessee at Knoxville

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Authors: Jonathan E. Towle

Christopher T. Clotfelter

Approved by: Dr. Gurminder Singh
Thesis Advisor

Arijit Das
Co-Advisor

Dr. Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current mobile devices are much more than the limited modality communication tools or digital assistants they were only a few years ago; instead they offer a range of content capture capabilities, including high resolution photos, videos and sound recordings. Their communication modalities and processing power have also evolved significantly. Modern mobile devices are very capable platforms, many surpassing their desktop cousins only a few years removed. TwiddleNet is a distributed architecture of personal servers that harnesses the power of these mobile devices, enabling real time information dissemination and file sharing of multiple data types from commercial-off-the-shelf platforms.

This thesis focuses on two specific issues of the TwiddleNet design; metadata tagging and data dissemination. Through a combination of automatically generated and user input metadata tag values, TwiddleNet users can locate files across participating devices. Metaphor appropriate custom tags can be added as needed to insure efficient, rich and successful file searches. Intelligent data dissemination algorithms provide context sensitive governance to the file transfer scheme. Smart dissemination reconciles device and operational states with the amount of requested data and content to send, enabling providers to meet their most pressing needs, whether that is continuing to generate content or servicing requests.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. OBJECTIVES 3
B. SCENARIOS 4
1. Disaster Response 4
2. Military Applications 6
C. RESEARCH QUESTIONS 7
D. SCOPE AND METHODOLOGY 8
E. ORGANIZATION 8

II. BACKGROUND 11
A. METADATA, TAGGING AND SYNDICATION 11
1. General Metadata Issues 12
2. Syndication and TwiddleNet 14
3. Tagging Issues Specific to the TwiddleNet Context 15
B. DATA DISSEMINATION 17
1. Data Dissemination Models 17
2. Peer-To-Peer Architectures 20
3. Issues Facing Data Dissemination in a Mobile Environment 21
4. Data Dissemination Issues Specific to TwiddleNet 23

III. TWIDDLENET OVERVIEW 25
A. CONCEPT 25
B. CONTENT TAGGING AND MOBILE DEVICES 26
C. PROTOCOL AND LANGUAGE 27
D. ARCHITECTURE 28
1. Device Specifications 28
2. Infrastructure 28
E. CLIENT AND TWIDDLENET CONTENT PRODUCER 29
F. TWIDDLENET PORTAL 32
1. Mobile Device Implementation 33
G. MODALITIES 33
1. TwiddleNet Operability Obstacles 33
2. Modality Interoperability 35

IV. TAGGING 39
A. PROVISIONING 39
B. METADATA COHERENCE 41
C. METADATA LIFECYCLE PHASES (*WHEN*) 42
D. METADATA MANDATE (*AUTHORITY*) 44
E. METADATA GENERATION METHODOLOGY (*HOW*) 46
F. SYNDICATION 47
G. ADD, DELETE, MODIFY 48
H. METADATA GENERATION PROCESS FLOW 49

1. Step One: Provisioning Document Setup 49
2. Step Two: Content Generation 49
3. Step Three: OnGeneration Tagging and Metadata Evolution 50
4. Step Four: Metadata Finalization and Transmission 51

V. DATA DISTRIBUTION 53
A. DISTRIBUTION TECHNIQUES 53
1. Posting Content 53
2. Viewing Shared Documents 54
B. RESOURCE MANAGEMENT TECHNIQUES 57
C. SENDING OCCASION OPTIONS 58

VI. SUMMARY AND CONCLUSIONS 61
A. FUTURE WORK 61
1. Portal Caching and Intelligent File Servicing 62
2. Syndication 63
3. Subnets and Handoff 63
4. File Transfer 64
5. System Log In 64
6. Security 65
7. Single Point of Failure 65

APPENDIX 67

LIST OF REFERENCES 69

INITIAL DISTRIBUTION LIST 71

LIST OF FIGURES

Figure 1. Power breakdown for a mobile device. From [1] 22
Figure 2. TwiddleNet Architecture Block Diagram 26
Figure 3. Generalized TwiddleNet infrastructure 29
Figure 4. Main TwiddleNet User Interface 32
Figure 5. Accessibility obstacles to TwiddleNet 34
Figure 6. Addressability obstacles to TwiddleNet 35
Figure 7. Peer-to-peer connection on same subnet 36
Figure 8. Sample Tags 41
Figure 9. Provisioning Document and Generated Metadata 41
Figure 10. Regular expression controlling inputs of type email 42
Figure 11. Predefined, on generation, and modify interfaces 43
Figure 12. Automatic tag selection interface 45
Figure 13. Mandatory contextual tags (predefined) 45
Figure 14. Sample Atom entry resulting from contained photo capture. 47
Figure 15. Provisioning document set up 49
Figure 16. Metadata file creation based on shared photo 50
Figure 17. OnGeneration tag population and ongoing modification 50
Figure 18. Metadata finalization and transmission 51
Figure 19. Posting process state diagram. 54
Figure 20. Sample result from a TwiddleNet search request. 55
Figure 21. Pulling data from the portal and mobile server. 55
Figure 22. Automatic update from portal. 56
Figure 23. Alert push and data pull sequence 57

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. TwiddleNet Test Devices 28
Table 2. TwiddleNet operability table 37

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I (Jonathan E. Towle) would like to thank Rich Betancourt and Rob Myers for their tremendous amount of effort devoted to the TwiddleNet project, without your help TwiddleNet would not be where it is today. More importantly, I would like to thank my wife, Sarah, for her continued support and patience during the late nights and short weekends.

I (Chris Clotfelter) would like to thank my wife and son for being so tolerant of the time I spent at school studying and writing. I would like to thank Richard Betancourt and Rob Myers as well; they were key in the development of the project, and I appreciate their help.

Finally, both of us would like to thank Professor Gurminder Singh and Arijit Das for their guidance, insight and inspiration.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Possession of a mobile device, whether it is a cell phone, smart phone, or personal digital assistant (PDA), has become the norm in many societies. The use of mobile devices has grown so much that not owning a mobile device is considered unusual. Yet, what is not so readily apparent is the ever increasing flexibility these devices offer to their users. The current mobile devices are much more than the limited modality communication tools or digital assistants they were a few years ago; instead they offer a range of content capture capabilities, including high resolution photos, videos and sound recordings. Their communication modalities have also evolved significantly. Mobile devices may frequently offer multiple communication modalities including Bluetooth, Wi-Fi, or digital data link. Further, the processing capabilities of these platforms continue to improve, surpassing desktop processors only a few years removed. Complex tasks not traditionally thought of as within the forte of mobility are now attainable and modern mobility is poised to offer true distributed computing

Media file sharing (photographs, movies, etc.) is a burgeoning social phenomenon in which participants can upload their content to third party servers to be shared via their websites. FlickR and YouTube are examples of this trend; proprietary media is uploaded to the respective server with accompanying descriptive tags, and that media is then made available to the public at large. While this model provides great data dissemination, it also has several drawbacks. First, providing media to these services often deprives the creator of ownership as part of the host agreement. So while their content might be widely consumed, there is no benefit conferred to the generator. Second, dealing with hosted services requires users to capture content and then upload it. This takes time and effort and leads to delay in dissemination of content/information.

Given the direction of these trends, it is reasonable to assume that content generated exclusively by mobile devices will increasingly appear in these shared forums. Nevertheless, the web-service model for content sharing is out of sync

with the distributed nature of modern mobility. The computational power of most mobile devices and their broad communication modalities can allow them to host their own shared content on resident personal servers, without the need for content upload to a third party service. This model allows content creators to share content without relinquishing ownership and can also provide subscribers with instant content alert.

The concept of mobile device file sharing is not without its own set of difficulties stemming from resource limitations inherent to the platform and the transmission medium. The most limiting factors in this area are power and bandwidth; mobile devices must always be cognizant of power consumption, and available bandwidth strictly governs the amount of traffic that a device can send and receive. Any mobile file sharing system must observe these limitations and employ some means of conserving these resources. One method to conserve bandwidth is to minimize the number and size of necessary transmissions to locate and acquire content; power can be managed by choosing when and how to conduct these transmissions.

In the context of a file sharing infrastructure, these goals may be achieved by using metadata to represent generated content. This metadata can include sufficient detail to describe the shared file, its size, residence, and other information pertinent to requesting clients. Content represented in this manner may be searched for with the same effectiveness as the actual file, but bandwidth demands to send and receive metadata are substantially less. So exchanges between personal servers and requesting clients can provide fully detailed information prior to providing the actual content, while minimizing bandwidth needs.

An intelligent data dissemination scheme can effectively manage power demands of this type of architecture. The nature of mobility presents special challenges in this regard. Depending on the device state, its remaining power and available bandwidth, providers may sometimes have to choose between providing content, providing metadata, or simply being able to remain operative.

Smart dissemination reconciles device and operational states with the amount of requested data and content to send, enabling providers to meet their most pressing needs, whether that is continuing to generate content or servicing requests.

A. OBJECTIVES

The objective of this thesis is to develop tagging and data dissemination parts of a distributed mobile server network, called TwiddleNet. TwiddleNet is designed to incorporate the abilities of mobile devices to perform real-time content capture and publishing, maintain full owner control of the content and allow users to subscribe, search, view and download content from other members in a peer-to-peer (P2P) fashion. The name was chosen to reflect the observance that the computational power of most mobile devices spends its time "twiddling" away in your pocket.

A key consideration in the development of the TwiddleNet architecture was the context in which it would be employed. Mobility presents special concerns that are not present in traditional computing; for instance, communication modalities are more varied in the mobile world. Devices commonly have Bluetooth, Wi-Fi or datalink access, or some combination thereof. Power is also a concern; a file sharing architecture designed for mobile use must address these issues. The TwiddleNet concept approaches these constraints from two distinct directions.

One design feature addressing the TwiddleNet mobile context is the implementation of shared file metadata. In traditional file sharing architectures, shared content in its entirety is uploaded to a third party server. Conversely, TwiddleNet provides only metadata to a central portal. This minimizes bandwidth demands by providing thorough descriptive information without the need for uploading content in its entirety. Once reaching the portal these metadata tags can form a networked file architecture. File information can then be searched in

a non-hierarchical fashion, enabling faster and more directed search and consequently improving network bandwidth efficiency. The tagging scheme will allow the user to share the maximum amount of information about a document while using the minimal amount of both bandwidth and power.

A second design feature is the development of algorithms that allow the user to maximize the device's power assets without limiting the user's freedom to use the application as he or she wishes. This scheme should take into consideration the current state of the device; is the battery fully charged or is it almost dead? Is there a connection available and if so, what signal strength is available? The algorithm should also be adjustable to the users' preferences given there current situation; does the user wish to share a photo as soon as he takes it or does he want to wait until he has taken several and send them all at once?

B. SCENARIOS

The TwiddleNet application can be used in many different scenarios such as social networking, law enforcement and military applications, emergency and disaster response and business applications. We designed TwiddleNet with two primary scenarios in mind. These two scenarios are disaster response and small unit military applications.

1. Disaster Response

Jon is the head of a disaster response team that is responding to a small town that has just been hit by a hurricane. The team of first responders includes local law enforcement agencies, fire departments, emergency medical personnel, utility workers and engineering experts, each of whom are equipped with a WiFi enabled smart phone. Jon and his team have deployed a system of portable, battery operated WiFi access points throughout the effected area. The team of first responders is dispersed throughout the disaster area assessing damage.

Bob is a utility worker assessing the damage to the power lines on the west side of town. As he is traveling to an area without power he spots a washed out road. Bob takes several pictures of the road damage, labels them and shares them on TwiddleNet. The TwiddleNet application automatically tags the images with date/time, location and author information. Jon is heading up the command post and gets a notification that new pictures are available of road damage. Sarah, head of the infrastructure assessment team, also gets a notification and can make an initial assessment of the damage and dispatch a repair team to the location or place it lower on a prioritized list, all in real time based on several digital photos.

As a first responder to a natural disaster such as a hurricane, tornado or earthquake, you need current, real time information in order to asses the damage and properly allocate assets. With TwiddleNet in place and all field representatives members of the network armed with a smart phone/PDA, information would be able to flow freely from member to member and from members to command post. Given the hurricane response scenario above, there will be extensive damage spread over a large area. This damage would possibly include human casualties, structural damage to residential and commercial buildings, infrastructure damage to roads, utilities and levees, and communications system damage to telephone and cellular structures. Using TwiddleNet, the responding agencies can deploy portable, battery powered wireless access points using 802.11 to establish the network. Once the TwiddleNet network is in place, field representatives from the different responding agencies can begin to take and share pictures of damage from their area of responsibility. Since TwiddleNet allows the user to label the images with summaries and key words any other user could search the TwiddleNet data base via keywords such as flooding, and immediately see the images from the field. TwiddleNet can also be used to send notices to subscribing clients when new pictures of interest become available. If a user were to subscribe to images that pertain to flooding then they would automatically receive a notification on their smart phone that new images were available and would be able to view them

immediately. Now multiply this ability by the number of personnel in the field (there could be hundreds). A command post would receive real-time information on damage assessments throughout the entire area of interest. This real time data would allow the decision makers to prioritize and allocate emergency and recovery assets to the proper areas with minimal delay.

2. Military Applications

Alpha Company is tasked with patrolling a five block urban battle zone. They have subdivided the area in to two areas with day and night patrols, each consisting of a squad size element. Each squad is equipped with several smart phones and a vehicle mounted wireless access point. Prior to the mornings patrol, the squad leaders review previous patrols imagery of the area. During the patrol the squad takes photos of the current situation, to include several suspicious men. Alpha Companies command post and intelligence shop receive instant notifications of the new images and can now compare the photos of the suspects against a database of known and suspected insurgents. As the patrol continues to take photos at pre-designated check points the command post is able to track their progress and monitor the larger situation by observing numerous patrols at once. For instance, they would be observing the behavior of the crowds or lack there of.

Distributed operations call for greater dispersion, smaller teams, and flexible communications packages. In spite of this, most units do not have the equipment or enough trained radio operators to provide reliable communications if the dispersion is over too great an area. For example, the table of equipment and the table of organization for an infantry platoon do not provide enough radios and trained radio operators to allow proper command and control (C2) if the unit operates as elements smaller than a squad beyond a certain range. This capability gap can be filled with an Internet Protocol (IP) based wireless data architecture hosting a distributed data dissemination application. Stated simply, there are a multitude of handheld devices on the market that can communicate in three or more frequency bands. It would take only a few vehicle mounted base

stations to create a wireless IP based architecture to support these multi-band mobile devices. With the appropriate encryption, standing up this network would be no different than establishing any other radio net, but in this case the radios are WiFi or WiMax capable mobile devices.

This “net” could be used to augment intra-team/platoon communications as well as be used to transmit what every Combat Operations Center (COC) wants and cannot get fast enough; real-time imagery. Additionally, since the network is IP based, the right dispersion of base stations can allow a mobile device to reach any other device in range of a base station even if the devices are not in range of each other. This is far more flexible than any doctrinal VHF radio net that requires a dedicated re-transmission site to accomplish the same thing. However, this flexibility alone does not accomplish anything for a unit unless there are applications running on this “net” that facilitate C2. The TwiddleNet application, which can run on any wireless IP based network, is such an application. Comparable to the various chat applications that exploit IP network infrastructures, TwiddleNet can enable another type of rapid data dissemination over such a network.

C. RESEARCH QUESTIONS

1. How can an automated tagging scheme be implemented such that it minimizes interference with ongoing user generated data creation?
2. What syndication formats are currently in use? Identify the formats that are most suitable to the TwiddleNet concept and allow for standardized syndication and tagging.
3. With respect to the TwiddleNet architecture, what are the best techniques for achieving information distribution?

4. Given the limitations of mobile devices, when is the ideal time to transfer data from the device in regards to signal strength, power level and file size?
5. How does the current context of the device affect the devices ability to send and receive data? I.e., is there a connection? Is the device's connection currently being used?

D. SCOPE AND METHODOLOGY

The TwiddleNet system is designed to allow real-time file sharing between mobile devices. Interface methodology was tailored to that end, within the governance of these qualitative principals: 1) Generated content should be quickly and easily tagged. 2) Tagging should not interfere with ongoing content generation. 3) Offered metadata sets must be readily expandable; related device provisioning should be minimized. Device sensitive methodology was also employed, with the goal of optimizing resources within the context of a potentially aggressive sharing environment. Data dissemination algorithms must maximize device resources while not severely curtailing file servicing.

The scope of this thesis will include an investigation of mobile servers, metadata generation and file sharing models.

E. ORGANIZATION

The chapters in this thesis are arranged according to the following topics. Chapter II is a review of previous work related to mobile servers, metadata generation, and file sharing models. Chapter III is an overview of the TwiddleNet system. This chapter will describe the TwiddleNet concept, detail the system architecture and discuss aspects of the application software development. Chapter IV will detail the metadata collection process. This chapter will discuss general issues associated with metadata generation and tagging, and how those

issues were addressed in the TwiddleNet application. Chapter V deals with the data distribution process. This chapter will discuss device limitations and how those limitations were approached in the context of metadata and content distribution. Chapter VI completes this thesis with overarching conclusions and recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This section provides an introduction to metadata, tagging and mobile data dissemination and issues related to these topics. Additional considerations specific to the TwiddleNet architecture and its leveraging of these concepts will also be addressed, as will strategies applied to mitigate these concerns.

A. METADATA, TAGGING AND SYNDICATION

Metadata is data that describes data. Consider a human being as a type of data, for instance. Certain characteristics of that person are simple facts, such as their height, weight, hair color, fingerprint pattern or eye color. These characteristics can be described for all human beings and may be regarded as fundamental metadata. Nevertheless, other descriptors that may be accurately applied to that person are not so easily classified. Where is that person currently located? How hungry are they? What is their current emotional state? Data such as these may be considered as contextual; that is, these data vary depending on the context in which that human is currently found. Low level data is referred to as syntactic data and is often easily automated. Higher level contextual data is referred to as semantic data, and is frequently provided via manual user input [2].

Tagging is the process of assigning metadata to an object. Again consider a human being as the data object of interest. Recording the data previously enumerated onto an id card is a similar process to logical object tagging; descriptive metadata has been assembled and recorded on a single medium that is assigned to that person. This metadata can either be manually or automatically generated, and may follow a well defined vocabulary or tag provider group. In other instances, such as social websites, tags may not follow a defined vocabulary or may be provided by visitors to the site. The latter type of scheme is called collaborative tagging [3] and may lead to meta noise, a situation

in which content searches become less efficient due to irrelevant or incorrect tagging. Collaborative tagging is beyond the scope of this thesis; our tags will either be automatically generated or will follow a predefined set of input rules, if not a specific vocabulary.

Syndication refers to the creation and offering of metadata feeds on websites. Feeds are collections of individually grouped metadata, each of which describes a specific piece of content hosted by the site. These feeds interact with many browsers and other software applications to provide automatic content download to subscribing clients. So if a feed were offered on a particular website, then updated content would be periodically sent to subscribers automatically as part of browser interaction with the site. The two most widely accepted syndication formats are RSS (Really Simple Syndication or alternatively RDF Site Survey) and the Atom Publishing Format, or Atom for short [4, 5].

1. General Metadata Issues

One of the key strengths of using metadata with distributed information systems is its searchability. Metadata can readily be parsed into individual content descriptors and placed within a database, allowing content to be easily located. A collection of files then becomes a networked structure, in which data may be located based on some combination of descriptors. Searches in this type of system generally return result sets that more closely match what the user wants than searches in traditional folder based, or hierarchical, data organizational schemes.

Some methods of generating metadata can be problematic. For instance, user defined semantic metadata can sometimes produce incoherent data leading to meta noise and reducing search effectiveness. This problem can be manifest even in non-social, owner-driven tagging schemes. Consider a tag intended to describe the height of a photographed mountain; owner-driven responses could reasonably be “very”, “10000ft”, “6000mtr” or “!@#$”. But providing strict vocabularies can curtail metadata expressiveness. So even in owner-driven

tagging schemes, some means of guaranteeing meaningful tags that still retain expressive power must be employed.

Lack of metadata expressiveness is rooted in the difference between semantic and syntactic tags, a gulf referred to as the semantic gap [2]. Semantic data can be very naturally descriptive, but cannot easily provide numerous file specifics (size, resolution, and the like) without a great deal of effort. Without this amplifying information, a document cannot be fully described; while the user may have a good understanding of the content, he does not have sufficient information to determine if the file should be downloaded in the context of his current device state. Semantic data must also be useful; hair color, for instance, is not a useful tag heading when viewing an image of a vehicle. Conversely, files described only by syntactic information lack meaning [6]; a user can determine if the file is downloadable, but not if the file actually provides the desired information. Metadata must have elements of both syntactic and useful semantic tags to be relevant, meaningful and retain powerful expression [6, 7].

But user supplied metadata is often difficult to obtain. Users have generally responded unfavorably to tagging when manual input was required, either due to time limitations or to simple unwillingness [6, 8]. However, only applying syntactic tags at generation time and allowing for amplifying notations later on has been shown to lead to data erosion [6, 8]. By providing durable user defined tags prior to a generation session this effect could be mitigated somewhat. These contextual tags, manually entered and modifiable at any time, could be applied to all generated content. In this fashion, each piece of outgoing metadata would have a mixture of contextual and syntactic tags, allowing even very lean transmissions a baseline level of expressiveness. These durable tags also add relevance to the content, serving to minimize data erosion due to deferred tagging where the creator may need some hints as to the context in which the content was generated.

The perception of when metadata should be assigned is another issue in the tagging process. In terms of media content, automatic tags are most often

discussed as being provided at generation time; contextual tags are generally discussed in terms of generation time, or at some later time when tags for a batch of content might be populated in a single sitting. Yet, it is useful to think of tagging as an ongoing process, both from the semantic and syntactic perspectives. Content may then be described at appropriate times in its evolution, and descriptors can be changed or enhanced to reflect changes to the parent file. Metadata then becomes a living document, itself describing evolving content.

2. Syndication and TwiddleNet

Syndication feeds contain a collection of eXtensible Markup Language (XML) structures referred to as Atom entries. Each entry contains a set of prescribed XML tags. Browsers that support the Atom standard interpret the information presented in each entry to determine the described file particulars (its residence, size, when it was last updated) and the actions the browser must take. A “pod” refers to the content described in the Atom entry, commonly media (hence the derived term “podcasting”). Nevertheless, there is no restriction in the Atom standard as to the type of file that can be contained. Files are fetched based on the Uniform Resource Locator (URL) mandated by the standard to be part of the entry. Atom enabled browsers generally perform updates as prescribed by the feed to which they are subscribed; yet, updates can be sought manually at user demand.

Atom has several desirable features: 1) It is fully extensible. By applying a namespace to the metadata, administrators can freely apply non-Atom-standard tags. 2) It is based on Representational State Transfer (REST) [4, 5], e.g. its interfaces are defined using only XML and HyperText Transfer Protocol (HTTP). Metadata posting and retrieving can then be conducted using standard HTTP PUT, GET, POST and DELETE. 3) It supports podcasting of any file type. This provides a means to share files of all types, instead of only a limited subset. Incorporating syndication into the TwiddleNet design could provide a ready

solution widespread data outreach, without the need for proprietary code. Atom is the syndication standard to which TwiddleNet metadata is constructed.

3. Tagging Issues Specific to the TwiddleNet Context

Beyond metadata issues, content tagging within the TwiddleNet architecture presented additional concerns related to the mobile context and the design goals of rapid data dissemination and expressive flexibility.

Usability and resource (power and bandwidth) preservation are two facets of the mobile context that can be affected by tagging. Usability issues stem from device form factor and input modalities and their effect on content generation in the given environment. This problem can be addressed in the following ways: 1) User defined tags can be entered at any time during the content generation process, so tags need not be applied immediately after content generation, and 2) By combining a minimal interface with an administratively flexible number of mandated user defined tags. Tag inputs can then be made on a simple-to-use interface and their number minimized according to mission needs. Bandwidth conservation takes place primarily through the use of representative metadata, since these transmissions are much smaller than the content they represent. Metadata entries can be further minimized by stripping any empty fields prior to transmitting the data.

Metadata timeliness is a key concept of the TwiddleNet development. Metadata is not frequently considered from the perspective of urgency. However, the TwiddleNet use cases are time sensitive and require that updated content be made available as quickly as possible. Content timeliness is first achieved by only uploading metadata. This saves time otherwise spent uploading data to a central repository. Time is further conserved by providing alerts of new updates to TwiddleNet members. This prevents lost time

corresponding to in-between periods of client browsing of the TwiddleNet portal; notification of new postings is instead pushed to the client as soon as the portal becomes aware of content availability.

Another tagging issue specific to TwiddleNet is the need to deploy the system quickly across a variety of usage modes and produce metadata that is fitting for the particular use (e.g. first responder, disaster relief, military unit). Two requirements are implicit in this design criterion: 1) required device provisioning to effect such a change must be minimized, and 2) the application code must be written such that changes to the code itself are not required. Such a flexible tagging model immediately appears similar to a collaborative scheme in which tags are generally uncontrolled; nevertheless, the need for reliable searchability prevents such an approach. Achieving the goals of a readily searchable networked data structure, minimized provisioning and unchanging code required an application that could produce proper tag offerings solely based on a single provisioning document.

Finally, the TwiddleNet design requires a tagging system that allows metadata to be easily modifiable. Note that this goes beyond simply populating empty tags at a later time. Since constructed metadata would be transmitted at the first available opportunity, then future modifications must necessarily be reflected both in the device and in the portal offerings. So changes to described content or simply manual amplifications would exact a cost in bandwidth and consequently in power. Minimizing bandwidth costs, whether from changes to content or to the actual metadata, demanded an application that could manage outgoing metadata effectively, and could reconcile metadata changes based on what had already been transmitted from the device and what had not. Such an approach would satisfy the need for readily growable metadata, while keeping bandwidth usage to a minimum.

B. DATA DISSEMINATION

The trend towards ubiquitous computing, where users are able to access data in any location at any time, increases the challenges faced by mobile computing. Data dissemination within today's mobile environment requires actively accessing rapidly changing, dynamic information and therefore requires different techniques than the traditional client-server model. The client server model, where users request specific data from a server when they need it, lacks the ability to efficiently provide timely and up-to-date information to consumers. Mobile Data dissemination is additionally made challenging by the inherent limitations of today's mobile devices. Mobile devices are limited in power and bandwidth, have limited coverage and are subject to inconsistent connection quality.

Data dissemination in its basic form can be defined as the transfer of data from producers to interested consumers, and it has three objectives:

- Make new content available
- Update existing content
- Remove out of date content [9].

1. Data Dissemination Models

Data Dissemination techniques can be broken into two major categories: push and pull methods. Push methods require that communications initiate with the source while pull methods require that communications initiate with the consumer. Furthermore, these methods can be broken down into two forms called periodic and aperiodic. Periodic communication is defined as communications that occur on a regular basis such as a timed interval. Aperiodic communication is defined as communications that have no predefined interval or

time for information transfer. These methods can be combined to form four basic data dissemination techniques; aperiodic pull, periodic pull, periodic push and aperiodic push.

The aperiodic pull model is the model that dominates information dissemination on the internet today and is better known as the client/server model. Information is transferred if and only when a client requests data from a server. The server then responds by sending only the requested information to the client. This continues until the client has received all of the information that he needs. In this model the server sits idle until it receives a request from a client. These requests are sporadic and do not follow a regular interval. The benefits of aperiodic pull are that it is a simple model and it does not require sophisticated programming or any upgrade to existing infrastructures. On the other hand, aperiodic pull does not preserve the timeliness of data, and requires the user to repetitively request for information which does not spend the user's time wisely.

Periodic pull, also referred to as smart pull or polling, decreases the reliance on the user to request data and furthermore increases the timeliness of the data delivery to the user. In periodic pull, clients are configured to automatically send requests to a client on a regular basis [10]. These requests are for specific information of interest such as sports scores, stock quotes or traffic updates. Polling is an improvement over aperiodic pull in that the automated requests relieve the user from the burden of always requesting data from the server. This in turn improves the timeliness of the delivery of the data and therefore provides more up-to-date information. The benefits of polling are that it maintains the simple architecture of the client/server model, and it increases the timeliness of data transmissions. In spite of this, data can still become stagnant if the request interval is too long. Another disadvantage of polling is that it wastes resources such as bandwidth and device power. For example, most data does not change very often but it is important to know when

it changes. To receive up-to-date data, the client must request data from the server on a small time interval. This results in many connections and requests that receive no new information.

Push implementations, also called content-based publish/subscribe architectures, differ from pull methods. In a publish/subscribe architecture, information is produced or collected by a publisher (server). When new information is available the publisher will issue a notification to a client. Clients are required to place standing orders or subscriptions for the specific information that they are interested in. Subscriptions then act in a Boolean fashion to determine whether notifications match the consumer's request. If evaluated true, the publisher will forward the notification to the subscriber.

Periodic push increases the efficiency of the dissemination model by completely relieving the user from the task of requesting data by broadcasting data of interest to users as soon as new information is available. However, it still wastes a considerable amount of resources by potentially broadcasting data that the users already have. The most efficient information dissemination model is aperiodic push. This model makes the most out of the available resources by only having the publisher send a notification when new data is available. Furthermore, aperiodic push offers the following advantages for real time data dissemination:

- The information source can initiate communications if and when data has been created or has changed.
- A minimum of one message is required to complete the transaction.
- Multicast communication can be implemented [9].
- Users are able to view previously downloaded information faster than data that is downloaded on demand.
- This paradigm matches with the hands free interfaces required in today's mobile device applications [10].

Despite the benefits, push methods do have their weaknesses. Typical push methods employ a one-to-many scheme where the publisher delivers content directly to each subscriber. This leads to limitations in the scalability of the network. Push methods also require more complexity in the form of content filters and “smarter” servers, to achieve an acceptable quality of service [11].

2. Peer-To-Peer Architectures

Peer-to-peer (P2P) architectures have many benefits to offer mobile data dissemination. By using a P2P architecture the limited bandwidth, power and device storage of one device can be massed together to create a more powerful resource. P2P allows devices to conserve bandwidth by reducing the amount of data transferred over a network. Reduced data transmissions in turn reduce power consumption. Finally, since files can be shared between peers in P2P architectures, individual device storage capacity is conserved.

There are many advantages that P2P architectures have over centralized client-server architectures such as reduced censorship, increased accessibility of popular content, less susceptibility to single point failures and increased privacy. Many of these issues can be witnessed in the Web today. For example, popular content residing on a single server can actually become more difficult to access as demand increases and floods the server. Similarly, censorship is difficult to prevent due to the centralized nature of a server. Centralized architectures are also very susceptible to technical failures, if a server fails or goes off line data is no longer accessible to clients. One of the most troubling weaknesses of the centralized model is lack of privacy. Currently there are companies, such as DoubleClick, Inc., that gather large amounts of user information from multiple servers and data bases and cross reference the information to gain detailed information about customers. This can all be done without customer knowledge or permission [12]. All of these issues can be addressed with P2P architectures.

P2P is defined by R. Scholmeier as a network where “participants share a part of their own hardware resources. These shared resources are necessary to

provide the service and content offered by the network. They are accessible by other peers directly, without passing intermediary entities." There are two distinct forms of P2P networks: pure P2P and hybrid P2P. In pure P2P all network entities are equal and if any randomly chosen entity were removed the network would not suffer any negative consequence. Hybrid P2P requires there to be a central entity that uniquely provides part of the network functionality [13]. Napster is an example of a hybrid P2P architecture where MP3 files are shared among peers who use a central server to locate the peer that owns the files of interest. Once the peer is identified a direct connection would be made between peers. Likewise, Freenet is an example of a pure P2P architecture where client and server are identical with no centralization required [12].

3. Issues Facing Data Dissemination in a Mobile Environment

Mobile device technology has grown exponentially in the past and continues to grow at a rapid rate. Yet, applications and device functions are also becoming more complex and require more resources from the device and network. Limitations that restrict mobile applications are power, bandwidth, latency, inconsistent connectivity, sparse coverage and cost.

Power resources have always been an issue of concern for users of mobile devices. Many of the applications and functions of mobile devices consume relatively large amounts of power. This fact coupled with the limited amount of power that today's mobile device batteries can provide makes power management a critical issue. Some of the largest power consumers in mobile devices are the Wi-Fi and Bluetooth radios, screen back light, and processors. Pering et al. provide the following statistics for a mobile device that is connected and in idle mode with backlight and LCD off.

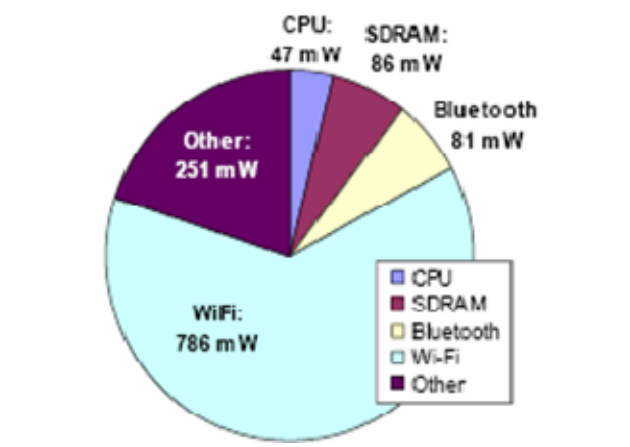

Figure 1. Power breakdown for a mobile device. From [1]

The bandwidth in mobile networks is very inconsistent. Speeds can vary from network to network and are dependant on the protocol in use. Bandwidth rates can vary from 9.6 Kbps in 2G networks to greater than 56 Kbps in 2.5G networks to even higher in 3G networks. Another important variable within wireless networks is the latency that users experience. Many times the delay that is experienced is blamed on poor bandwidth but in fact is caused by the network latency. Latency can be responsible for an additional 30 seconds in overall connection time [14].

A further limitation faced by mobile computing is the lack of a consistent connection. Mobility can cause frequent and unexpected breaks in network connections. This break in connection can be caused by a loss of connection due to a bad handoff from one access point to another, or could be due to being out of range from an access point. Lack of network coverage is also problematic for ubiquitous computing. Typically, coverage is greater in highly-populated areas such as metropolitan cities. However, this does not guarantee a connection as many physical barriers such as buildings and tunnels can block the signal.

4. Data Dissemination Issues Specific to TwiddleNet

Data dissemination techniques can be used to maximize the power resources available to a mobile device. This can be done by minimizing the time that the device spends sending and receiving updates and notifications over its wireless connection. Data dissemination techniques can also improve power conservation by considering the context in which the user is generating content. The user may be generating urgent content and may want to make this content available as soon as possible. This would require the application to send an update as soon as a document is generated. On the other hand, the content might not be urgent and can wait for a more opportune time to be sent, such as when the user has generated numerous documents. Now the application can make one connection and send all the updates at once, conserving power and bandwidth. The flexibility to send updates either on document generation or collecting documents in order to send at a later time also gives the application the ability to adjust to inconsistent connections and limited network coverage. If the user is in an area with poor connectivity, the application can monitor the connection status and send whenever a connection is detected.

Receiving up-to-date information is also a high priority in the TwiddleNet architecture. The publish-subscribe model fits well with the TwiddleNet architecture, allowing the subscriber to receive up-to-date notifications as soon as possible. The issue is to determine which push technique is ideal for TwiddleNet. If the subscriber receives a notification and the document when new content becomes available, then the application is potentially wasting bandwidth and power if the user does not wish to view the content. If however, only a notification that new content is available is sent, then the user can determine whether to download the entire content or not.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TWIDDLENET OVERVIEW

This chapter will introduce the TwiddleNet system. Coverage will include concept, developmental considerations, component specifications and generalized architecture. Special issues related to device modality and reachability will also be addressed.

A. CONCEPT

Modern mobile devices have advanced remarkably in recent years. A hallmark of modern mobility is the wide range of content capture capabilities now becoming commonplace on many cell phones and PDA's. Examples include high resolution photos, videos and sound recordings. Some of the devices also come pre-equipped with location sensors, primarily GPS (Global Positioning System). Their communication modalities have also evolved significantly. Modern mobile devices frequently offer multiple communication modalities including GSM/CDMA, GPRS/EDGE, Bluetooth, and Wi-Fi. Further, the processing capabilities of these platforms continue to improve, surpassing desktop processors only a few years removed. While the devices are quite powerful, their capabilities are not being exploited fully. Much of the time these devices are sitting idle doing nothing. TwiddleNet aims to exploit these capabilities to extend their use.

TwiddleNet is a mobile personal server architecture that enables real time sharing of user generated content (UGC) across a network of distributed devices. This sharing is accomplished through the use of descriptive metadata tags that are assigned to a file once it has been shared. These metadata files are uploaded to a centralized portal and arranged for efficient UGC location and searching across the TwiddleNet mobile server network. UGC in this instance should not be construed simply as hardware driven media or sensor captures. TwiddleNet is capable of sharing UGC of any filetype within the capacity of the

device to produce. Combined with the power of modern mobile devices, the result is a truly distributed network that can readily share a wide variety of information in real time.

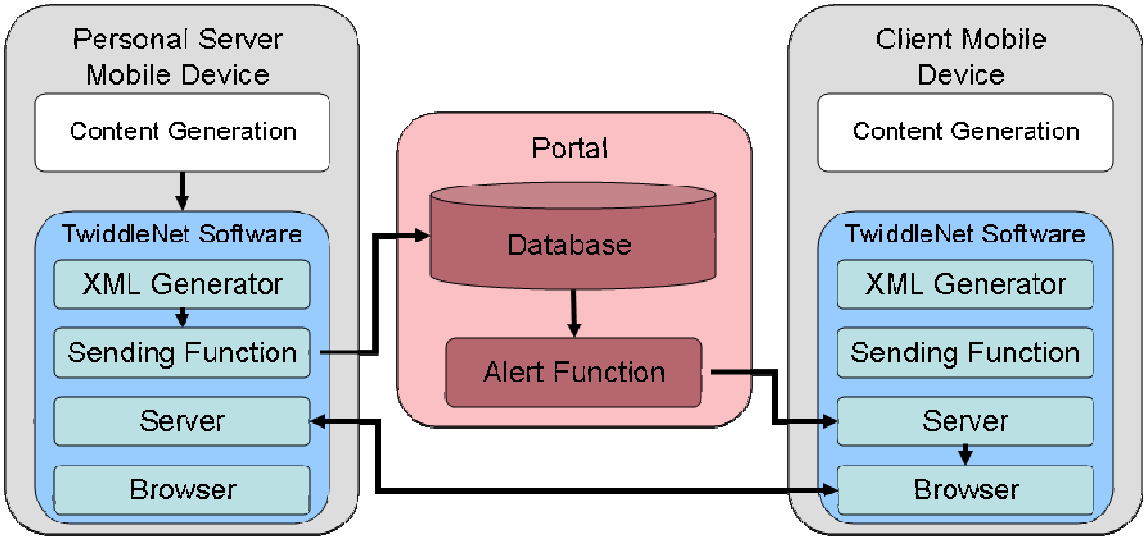

Figure 2. TwiddleNet Architecture Block Diagram

B. CONTENT TAGGING AND MOBILE DEVICES

Mobile file sharing assisted by metadata generation is not a new concept. In [15], third party sensor nodes were combined with mobile device photo generation to assemble automated contextual metadata. Associated metadata was strictly automated, no amplifying user input was affiliated with the file. The Mobile Media Metadata (MMM) prototype detailed in [8] applied low level contextual metadata to generated photos and allowed natural language inputs. The system also employed metadata sharing by use of a photo identification algorithm at a centralized server. In [16], automatic and manual entry metadata were combined under the notion of ontology to classify photos generated on mobile phones. In all of these approaches, both metadata and accompanying photo were uploaded to a central repository.

The TwiddleNet system uses manual entry and automatic metadata to classify files; yet, it differs from the previous approaches in several ways. First, TwiddleNet fully leverages the content capture capability of the device. File sharing in TwiddleNet is not limited to any particular type of shared file. Users may freely annotate and share any file of their choosing on the TwiddleNet network. TwiddleNet also does not upload UGC to a centralized repository; instead it employs resident personal servers to accomplish file sharing. Finally, TwiddleNet designs its metadata around the Atom syndication standard [5], expanding its reachabilty to an internet wide audience.

C. PROTOCOL AND LANGUAGE

A primary concern when developing the TwiddleNet system was the internet protocol that would be observed for file transfer. One of the most important considerations when making this decision is wide distribution of content. We want to be able to support a syndication technology as well as make content accessible to tertiary clients that may not be running the TwiddleNet application. Based on these criteria, we selected HTTP. HTTP is supported on most internet browsers, so it satisfies reachability concerns for devices outside of the TwiddleNet network; users can simply surf via their typical browser to access TwiddleNet content. The Atom syndication standard is REST (as explained in Chapter II) based, so associated metadata requirements and operations can be met with standard HTTP PUT, GET, POST and DELETE commands.

Another important decision is the system development language. While the TwiddleNet application can generally support all types of file transfer, its genesis is in the type of spontaneous UGC typical to media, i.e., taking photos, or recording audio and video. Preference was given to devices capable of generating these types of files; currently these are primarily smartphones or PDA's. Metadata handling also weighs in the decision; since Atom is the

syndication standard to which the application is designed, XML is the metadata vehicle. C# lends itself to both needs; a wide variety of UGC capable devices run Windows based operating systems (OS), and the .NET framework has well established XML tools. So, C# was selected to be the development language.

D. ARCHITECTURE

1. Device Specifications

A prototype of TwiddleNet is implemented using a variety of iPaq and other PDA's and smartphones. The applications are written in C#, and generally designed to be run on devices supporting Windows Mobile 5.0. However, the application was tested on devices running older Windows based operating systems without noticeable impact to the application's performance. The following table details devices on which the TwiddleNet application was deployed and tested.

Device	Operating System	Processor
hw6945	Microsoft Windows Mobile V 5.0	Intel PXA270-416MHz
h4155	Microsoft Pocket PC V 4.20.0	Intel PXA255-400MHz
h5550	Microsoft Pocket PC V 4.20.1081	Intel PXA255-400MHz
ppc6700	Microsoft Windows Mobile V 5.0	Intel PXA270-416MHz
Dell X51	Microsoft Windows Mobile V 5.0	Intel PXA270 624MHz

Table 1. TwiddleNet Test Devices

2. Infrastructure

A key concept in developing the TwiddleNet system is the notion of a distributed architecture; to the largest extent possible, data operations are pushed outward to participating devices. Consequently, TwiddleNet components must be capable of performing system functions on an individual device basis;

this results in a system having light weight components. There are only three roles in the design; the TwiddleNet portal, personal servers (PS, or content producers) and clients (C, or content consumers). The portal is the only component that has a centralized role within the architecture; producers and consumers behave independently, although their functions overlap to some extent. Personal servers are also functional consumers, but consumers may not have production capability, and in some cases may be external to the TwiddleNet system altogether. The general infrastructure of the TwiddleNet system is shown in the following figure.

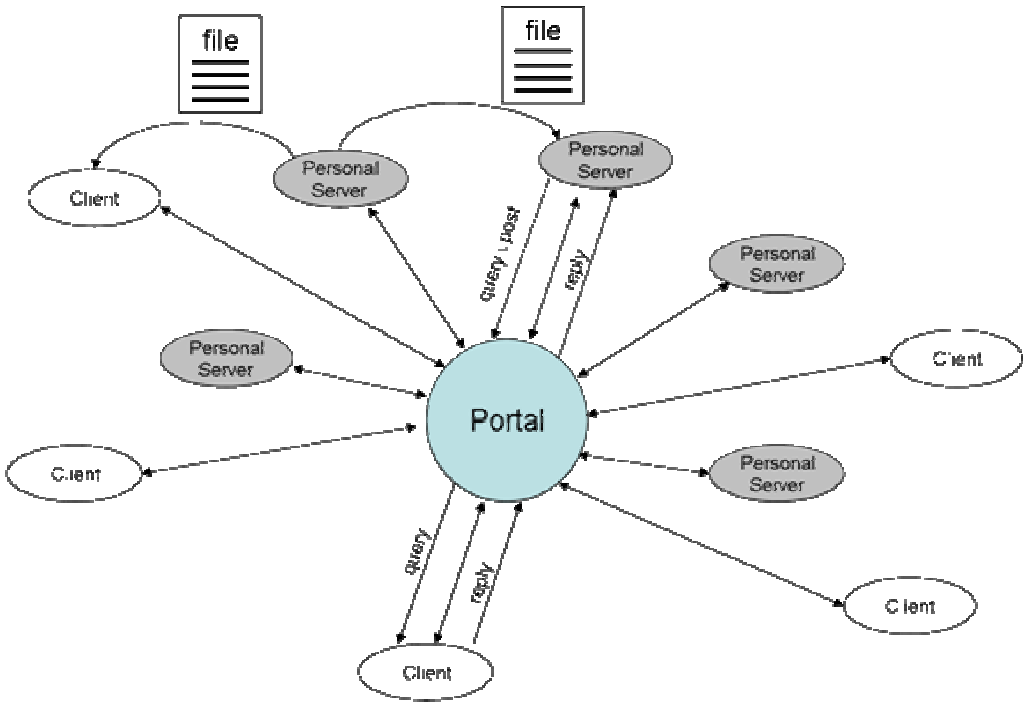

Figure 3. Generalized TwiddleNet infrastructure

E. CLIENT AND TWIDDLENET CONTENT PRODUCER

Mobile devices are key components within the TwiddleNet architecture, and have two fundamental roles; that of producer or personal server, and that of

consumer or client. Since TwiddleNet file transfer is HTTP based, the task of locating and downloading a file can be accomplished using a typical web browser. Users may access TwiddleNet content via the portal either using a separate web browser or from within the TwiddleNet application; the portal and participating devices draw no distinction. In short, the client role within the TwiddleNet system differs little from a client role in a traditional HTTP session.

The role as content producer is more complex. Content producers have several functions within the overall architecture. Fundamentally this includes creating UGC, creating and uploading associated file metadata to the portal and servicing requested files. These functions are further complicated by the mobile context. For instance, metadata upload and file servicing take place in a manner sensitive to the device's current state, so power level and available bandwidth influence application behavior. Content producers must also afford the user a convenient means to manage files; users should know what files are currently shared and should be able to change this state efficiently.

From the UGC creator perspective, the application should allow the device to leverage its content collection and creation capabilities. This includes software driven creation, as in spreadsheet or word processor generated files, as well as hardware driven collection capabilities. The latter would include all content generated from hardware native to the device (photos, audio or video files, or other sensor modes). There are many OS issues related to these hardware driven operations. Simply stated, hardware affiliated native applications across different devices vary in their interaction with the OS. This complicates the manner in which the shared directory is populated. In general these issues may be handled through the use of a timer, thereby allowing the OS to complete its handling of the generated file before conducting TwiddleNet operations, or by implementing a buffer directory in which OS operations will be completed prior to moving the file into the TwiddleNet directory space.

Metadata generation is another role of the content producer. This can be broken down into three subcategories: creation, upload and modification.

Metadata creation is driven by a provisioning document provided by system administrators and designed to reflect the usage environment. Tagging is accomplished with both automatic system input and manual entries provided by the user. Upload is the process of providing this input to the TwiddleNet portal. Note that this operation is not simply “create-and-push”; upload takes place as a consequence of user specified application options and device context awareness. Unless properly understood, this may sometimes be puzzling, as some users may incorrectly assume that the application is misbehaving when in truth some combination of states prevents transmission at that exact moment. Metadata modification refers to the application’s ability to persistently maintain and update metadata as UGC is altered or as associated metadata files are directly changed. An instance of the latter would include the case of a “Summary” tag that the user is able to further detail after the initial metadata has already been forwarded.

File servicing directly from the mobile device is a key aspect of TwiddleNet content producers. The mobile device server application is intended to be running at all times in the background of the device. The sole purpose of the server is to receive and respond to all legitimate file requests; both internally from other TwiddleNet devices, or from clients external to the TwiddleNet system. File requests come in the form of HTTP GET commands listing the particular file of interest. The server is capable of servicing more than one client at a time, increasing the robustness of the system.

File management is the final role of the content producer. Users need a simple mechanism to control the content they are currently sharing that is well integrated with the TwiddleNet system and the notion of ongoing content generation. This role is manifest in the basic TwiddleNet application user interface (UI). From this interface, users can control all file operations that influence actual content or associated metadata. This provides users the means to efficiently control their content within the generation context. The following figure displays the TwiddleNet UI.

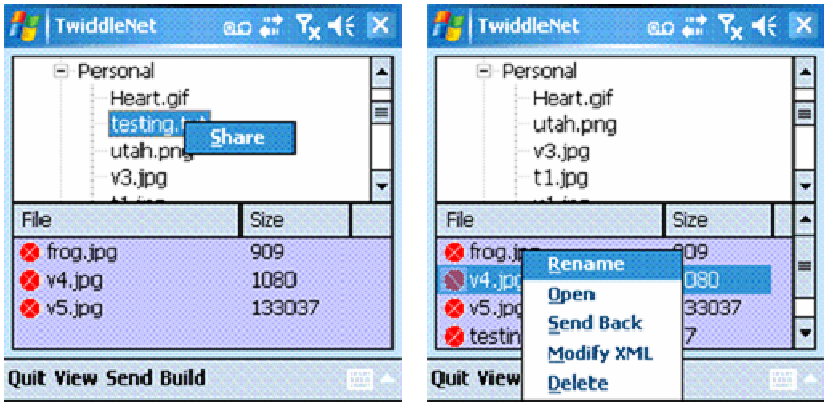

Figure 4. Main TwiddleNet User Interface

F. TWIDDLENET PORTAL

The TwiddleNet portal has three basic responsibilities within the TwiddleNet architecture. First, it must handle metadata posts from content producers. This involves accepting HTTP POST requests, parsing the metadata and populating a database with the received information. Second it must service file requests from enquiring parties, either from within the TwiddleNet architecture or external to it. This request would involve a database search producing a result set from which the user would select the desired file. The user would then be forwarded only a link to the owner device of that information. Finally the portal must alert subscribed users of newly posted content. This final function is currently limited to TwiddleNet content producers, and is not specific to any single keyword or subscription definition. When this function is activated, in our current implementation, all TwiddleNet content producers with active session receive pushed updates of new posts. A more general implementation will allow select users, who may subscribe by choice, to receive alerts.

1. Mobile Device Implementation

The TwiddleNet portal has been implemented both for traditional desktop computers as well as the same types of mobile devices as the TwiddleNet application. The mobile device implementation adds a great deal of flexibility and scalability to the baseline architecture, since infrastructure requirements often attached to larger computers are no longer necessary. Nevertheless, this does constrain this aspect of the architecture to the same resource limitations as on the application side; bandwidth, power, processing speed and memory all become considerations in this implementation.

G. MODALITIES

TwiddleNet is capable of operating on several separate network infrastructures, Wi-Fi on a Local or Wide Area Network (LAN or WAN), General Packet Radio Service (GPRS) on a Global System for Mobile communications (GSM) cellular phone network or Bluetooth on a Personal Area Network (PAN). This section discusses the operability of TwiddleNet within and between these environments.

1. TwiddleNet Operability Obstacles

There are many challenges to operating a web server on a mobile device. Currently since there is no significant demand for such services on mobile devices, network operators focus on optimizing network functionality and protecting network assets and customers. This has lead to network infrastructures that do not allow autonomous downlink traffic and provide poor quality uplink traffic [17]. The roots of the mobile server access problems can be divided into two categories; accessibility and addressability.

Accessibility refers to the issue of traffic reaching the mobile server. This is a common obstacle to mobile servers due to network operators employing

firewalls to block incoming traffic unless it is in the form of a response. This practice is widespread as it protects networks from malicious attacks.

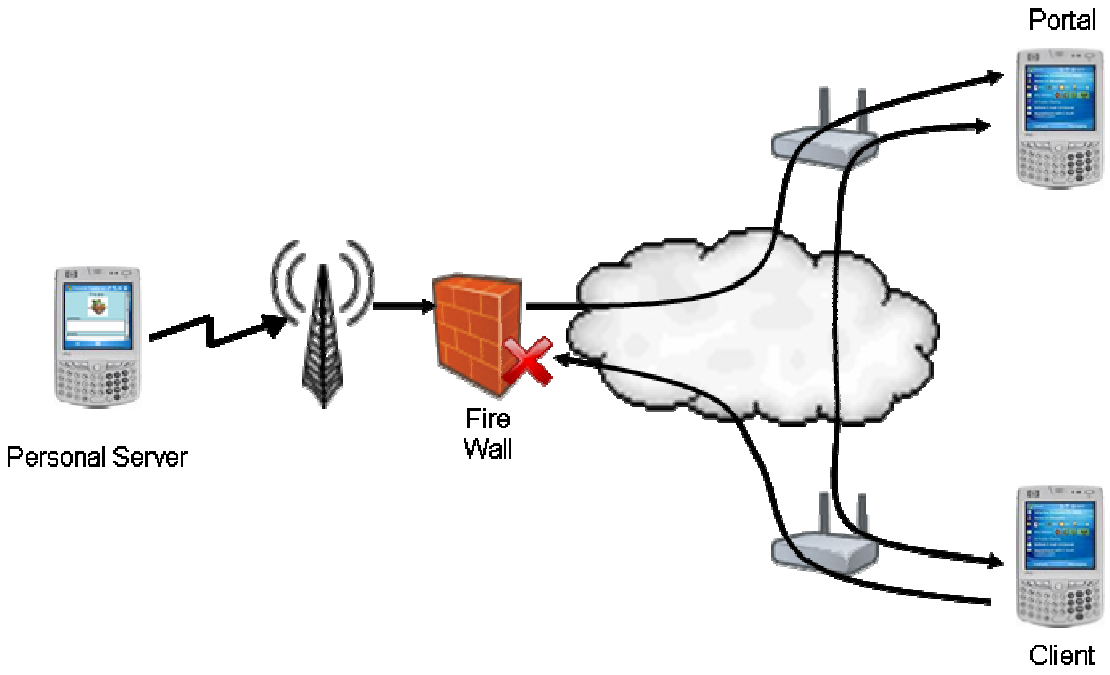

Figure 5. Accessibility obstacles to TwiddleNet

Addressability is a different issue that refers to the ability of a client to find the mobile server. There are two problems facing addressability. First, most mobile devices are assigned dynamic IP addresses, meaning a mobile server will constantly be assigned a different IP address. Second, most network operators implement Network Address Translation (NAT) to assign IP addresses within their network. While these addresses are defined within the network, they have no meaning to routers outside of the network [17].

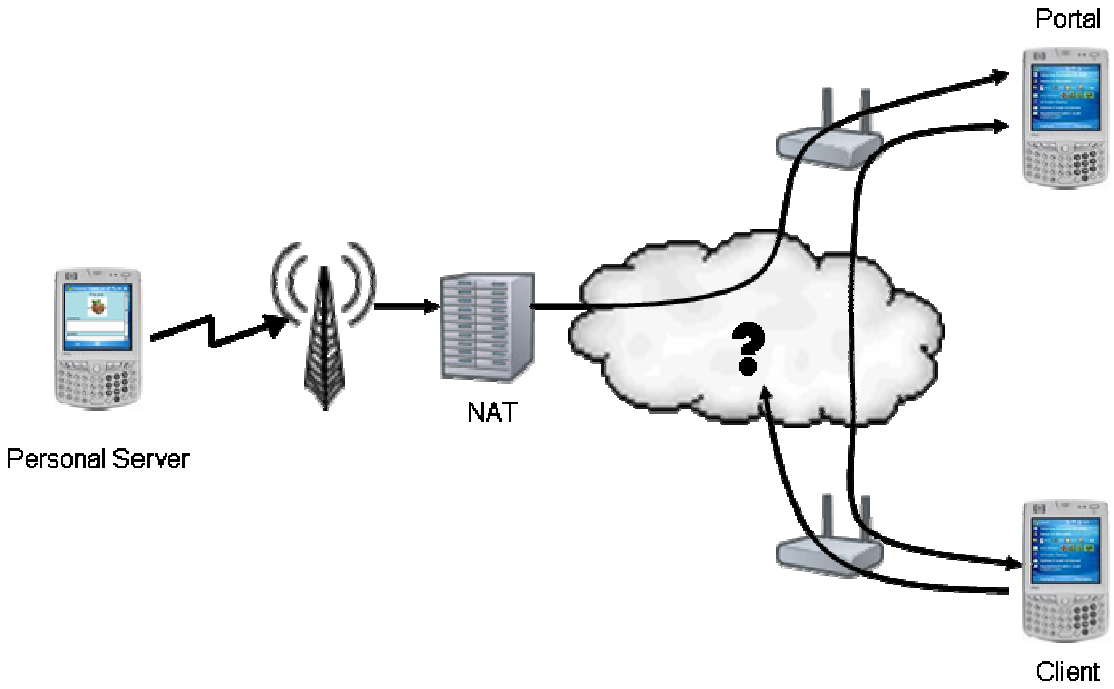

Figure 6. Addressability obstacles to TwiddleNet

2. Modality Interoperability

TwiddleNet has been tested over two different network infrastructures, GPRS and Wi-Fi, with mixed results. These varying results are due to network obstacles discussed above and the different configurations possible with the three basic components in TwiddleNet; portal, personal server and client. The different configurations of the TwiddleNet components are; all three in the same subnet, all three in separate subnets, or one of the components is in a separate subnet with or without firewalls or separate subnet employing NAT.

Mobile service providers place numerous restrictions on their networks to provide customer security and network optimization. Providers use both network address translators and firewalls, this prevents TwiddleNet from functioning properly when a personal server is on a different subnet than the client that is trying to reach it. First, due to NAT the address provided by the personal server would be meaningless to the client and therefore the client would be unable to

reach the personal server. Second, if the message were able to find the personal server, the message will be blocked by the firewall since it is a request message and not a response. On the other hand, if both the personal server and client are on the same GPRS subnet, even if the portal is not, TwiddleNet will function properly. This problem is not limited to GPRS networks. If the personal server and client are separated by a NAT and/or a firewall in a Wi-Fi network, TwiddleNet will similarly fail. The simplest solution to the obstacle is that the peer-to-peer connection must take place within the same subnet and not be blocked by NAT or a firewall. The more complex and general solution is to provide links among different subnets implemented by different mobile network service providers. This implementation has not been completed in this thesis.

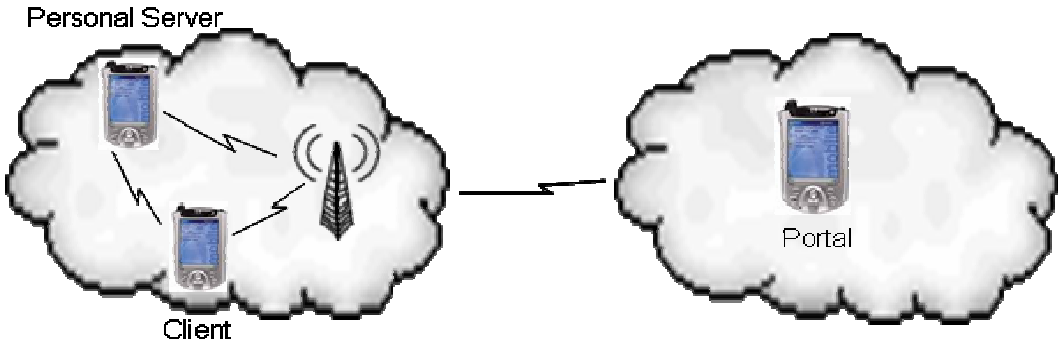

Figure 7. Peer-to-peer connection on same subnet

The following table displays the current modality combinations and their operability as it applies to TwiddleNet.

Network	Configuration	Functional
Wi-Fi	PS, C and Portal on same network	Yes
Wi-Fi	PS and C on same subnet with NAT and/or firewall Portal has dedicated IP address	Yes
Wi-Fi	PS and C on separate subnets with NAT and/or firewall Portal has dedicated IP address	No
Wi-Fi	PS, Portal and C on separate subnets with NAT and/or firewall	No
GPRS	PS, C and Portal on same network	Yes
GPRS	PS and C on same subnet with NAT and/or firewall Portal has dedicated IP address	Yes
GPRS	PS and C on same subnet Portal on a separate subnet with NAT and/or firewall	No
GPRS	PS, Portal and C on separate subnets with NAT and/or firewall	No
GPRS/ Wi-Fi	PS and Portal on Wi-Fi net W/O NAT and/or firewall C on GPRS connection	Yes
GPRS/ Wi-Fi	C and Portal on Wi-Fi net with NAT and/or firewall PS on GPRS connection	No
GPRS/ Wi-Fi	C and Portal on Wi-Fi net W/O NAT and/or firewall PS on GPRS connection	No

Table 2. TwiddleNet operability table

THIS PAGE INTENTIONALLY LEFT BLANK

IV. TAGGING

This chapter details the methodology related to the overall tagging process in TwiddleNet. Program flow is also discussed from content generation and onward to the point at which the tag file is prepared to be transmitted.

A. PROVISIONING

The TwiddleNet model requires that generated metadata be fitting and appropriate to the environment in which TwiddleNet would be deployed. First responders would have mission specific tags, as would military units, disaster relief, and so on. This should be accomplished without the need for modification and re-provisioning of the actual program code. These goals are achieved by generically writing the application such that a single configuration document can produce the desired tags.

Evaluation of the TwiddleNet tagging model demonstrates these grounding concepts: 1) a notion of lifecycle, or timing, needs to be applied to the provisioning document. This would facilitate ongoing content and metadata modification, as well as specify generation time and sending time tags. 2) Each tag should have a mandate, or authority, as to whether it will get populated. This will enforce minimal metadata generation, and will facilitate metadata documents that are tailored to system administrator needs. 3) How tags get populated be specified as either automated or user defined. 4) Type integrity of user entered tags be maintained. And, 5) the produced metadata should be compliant to the Atom syndication standard.

When, *authority* and *how* are the factors that determine the actual number of different tag types that can appear in the configuration document. The *when* factor has three categories; predefined (predefined), on generation (onGen) and on sending (onSend). *Authority* requires two categories (mandatory and optional) as does *how* (userdefined or automatic). The combination of these

attributes yields 12 possible tag types; however, for purposes of the TwiddleNet application two of these types will never be provisioned (userdefined/mandatory/onGen and userdefined/mandatory/onSend). Allowing these tag types would require user interaction with every metadata entry prior to its transmission. Such an arrangement would disrupt ongoing UGC creation and file sharing, hence their exclusion from provisioning documents. The application therefore has 10 possible tag types for provisioning document construction.

Based on these grounding criteria, we have selected XML as the means to express generated metadata. Each tag has a minimum of three descriptors (*when*, *authority*, *how*) that are expressed as XML attributes. TwiddleNet tags may also have further attributes, depending on their role within the application. Since the Atom standard [5] employs a REST approach (as explained in Chapter II) to web services, it defines its interfaces in terms of XML and HTTP. This allows tag extensibility through the employment of XML namespaces, while still remaining within the Atom standard.

The following figure presents two possible tag configurations, each demonstrating distinct tag types, namespaces and provisioning document construction. The three primary attributes (*when*, *authority*, *how*) in each configuration dictate how the tag is handled within the application. The first example details a *summary* tag that will be populated at generation time (*when* = “onGen”), by a manual user input (*how* = “userdefined”), but may or may not be utilized (*authority* = “optional”). Note that the *summary* tag is defined within the Atom namespace, so it does not require a namespace declaration. The second example details an *ipaddress* tag that will be populated at transmission (*when* = “onSend”), by an automatic function (*how* = “automatic”), but may or may not be selected for population (*authority* = “optional”). Note that the *ipaddress* tag is not within the Atom namespace, hence the namespace declaration in the opening brace (t:ipaddress). Both tags are unrestricted data types (indicated by *datatype* = “0”), so there is no regular expression restriction governing how they may be filled.

```
<summary when ="onGen" how="userdefined" authority="optional"
dataType ="0"></summary>

<t:ipaddress when ="onSend" how="automatic" selected="true"
authority="optional" dataType ="0"></t:ipaddress>
```

Figure 8. Sample Tags

A sample provisioning document is provided in the Appendix. For future reference, the following icons will be used in process flow illustrations to indicate the provisioning document itself and metadata documents describing generated content (below).

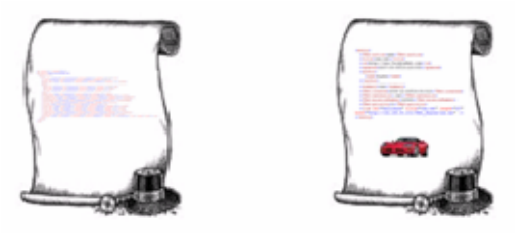

Figure 9. Provisioning Document and Generated Metadata

B. METADATA COHERENCE

TwiddleNet tags are only supplied by the content owner. While this generally serves to lower the incidence of metadata noise, it might still be insufficient to guarantee complete and coherent results for users who are locating content via portal searches. Insuring metadata consistency at the portal was not considered an option, since there would be additional bandwidth overhead associated with error detection and correction between the posting device and the portal. Such an implementation would also conflict with the distributed network design; to the largest extent possible, centralized data operations were reduced.

Consequently, metadata type integrity would need to be policed on the device side. The issues to balance were metadata expressiveness and type consistency. Users must have great latitude when providing input to maximize

contextual descriptiveness; nevertheless, those fields must meet criteria specified by system administrators via their provisioning document. This design feature is implemented through the use of a datatype attribute that indicates a regular expression against which a particular field will be checked. Inputs not satisfying those regular expressions are rejected. For instance, suppose a tag describing the size of an object required an integer as its input, a value of "very large" would be rejected, where "1000" would not. This is a very general example, but the concept can be extended to support proper names, date/time formats, and so on.

Figure 10. Regular expression controlling inputs of type email

C. METADATA LIFECYCLE PHASES (*WHEN*)

When developing TwiddleNet, it was useful to think of metadata generation in terms of shared file lifecycle. Unlike many metadata generation schemes, the TwiddleNet tagging model requires the flexibility to make metadata update dynamic and ongoing. This is considered critical in a system where amplifying information about formerly described content can continue to evolve or to become available.

TwiddleNet tags afford three options for specifying the *when* attribute of a tag; predefined, onGen, and onSend. These criteria specify the point in the lifetime of the metadata at which the value of a tag should get populated. Prior to initial access, a TwiddleNet provisioning document is completely unpopulated. This unpopulated state causes the application to offer a configuration interface in which the user enters values for predefined tags. Automated tags designated as predefined are also populated at this point; static owner data is applied here as is any data that must be applied to each shared file as part of Atom standard conformance. Resulting values are then retained in the provisioning document from that point forward. These tags are durable and their values will be applied to all future generated metadata.

Figure 11. Predefined, on generation, and modify interfaces

When new content is shared, an XML file associated with that content is created from the provisioning document. Automated tags designated as onGen are then filled, and in general an interface is presented to the user for population of contextual tags as well. However, manually entered contextual tagging may optionally be deferred until a later time, or not at all. This could be a simple user preference, or it could be a function of ongoing content generation to which assignment of metadata tags might be disruptive. OnSend tags are the final tags to be populated prior to transmitting the metadata file to the portal. These tags are provided in order to capture the most recent metadata possible associated with the file and the device state. Remaining device battery charge, IP address, or a GPS stamp are tags that would be well suited for transmission time population.

The ability to modify metadata at any time fills in the gaps between these distinct tagging states. Once a metadata entry has been created for a shared file, its tags thereafter become accessible for modification. Metadata can then be updated at any time while the associated data file is still in a shared mode. Changes to the described content or additional contextual details can be added to the metadata document as needed, completing the tagging process and fully encompassing the lifespan of the shared file.

D. METADATA MANDATE (*AUTHORITY*)

The provisioning document provides the baseline set of tags. Yet, there is no requirement that each and every suggested tag be a used when building metadata documents. This control is implemented through each tag's *how* XML attribute as given in the provisioning document. The implementation is straightforward; tags marked as mandatory must always be filled in, while tags marked as optional may be filled in at the user's discretion. Mandate designations apply to both contextual and automatic tags, but their implementation differs.

By default, automated tags will always get populated unless the user elects to preclude those tags from the generated metadata document. Figure 12 shows the user interface through which optional automatic tags are specified. The user simply selects the desired tags from those offered. Subsequent XML contains only those selected tags. In the case of contextual metadata, a tag's mandate is indicated by the presence of an asterisk on the input box label. Tags not having an asterisk are optional, and may go unpopulated. Contextual tag mandate is shown in Figure 13.

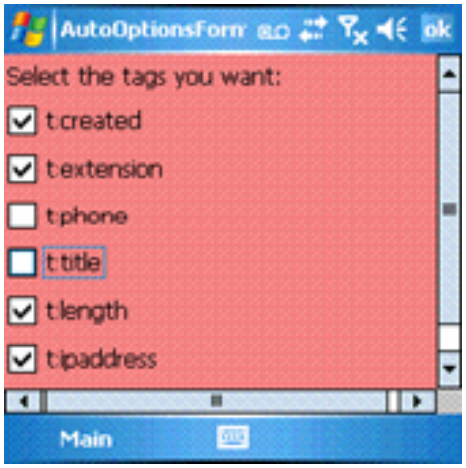

Figure 12. Automatic tag selection interface

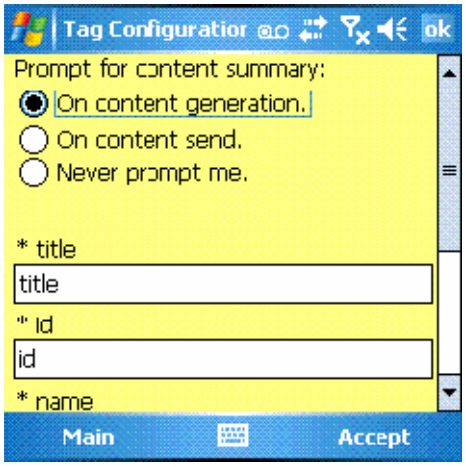

Figure 13. Mandatory contextual tags (predefined)

Elective tagging based on mandate serves several functions. The first is the facilitation of the mobile context in which TwiddleNet is used. Since not all tags must be filled in, leaner metadata can be sent to the portal. This allows users to be cognizant of their device resources and enables them to provide only the amount of information they deem necessary, thereby minimizing resource load. Secondly, the tag mandate enforces coherence to the Atom standard. The Atom standard requires certain tags to be included with each entry level node. By making these tags mandatory, resulting metadata will always contain the appropriate information for Atom compliance. Finally, optional contextual tags facilitate the content collection process; since tags need not be populated, the user can streamline tagging to the extent necessary according to the current environment.

E. METADATA GENERATION METHODOLOGY (*HOW*)

A tag's *how* attribute simply lets the application know how the tag should be processed. User-defined tags are presented in contextual tag user interfaces, while automated tag processing takes place without the user's knowledge. With each automated tag there is the issue of ensuring that corresponding application code is in place. Selection of an automated tag implies that a routine exists that can farm the desired metadata from some system resource. This issue constrains provisioning; automated tags cannot be placed within a provisioning document without ensuring that code exists to support those tags. The TwiddleNet application design mitigates this issue somewhat, since all data farming code is held within a single class. In the case where additional data farming methods are necessary, only this section of code will need modification.

F. SYNDICATION

TwiddleNet is predicated on the notion of widespread sharing of information in a rapid manner. Two approaches have been taken to achieve this dissemination; content push and content pull. Content push incorporates an alert mechanism that notifies subscribers of new content as it is posted. Content pull is accomplished via HTML and uses a typical web browser. In spite of this, the TwiddleNet device side application has been designed to produce valid Atom feeds as well. Leveraging this syndication technology could radically increase data dissemination.

```
<entry>
  <title>v4.jpg</title>
  <id>http://www.TwiddleNet.com</id>
  <updated>2007-03-05T13:23:00Z</updated>
  <author>
    <name>Alpha</name>
  </author>
  <t:created>2006-12-03T19:05:51Z</t:created>
  <t:extension>.jpg</t:extension>
  <link rel="enclosure" title="v4.jpg" length="1000"
        href="http://192.168.55.101/tnet/shared/v4.jpg"/>
</entry>
```

Figure 14. Sample Atom entry resulting from contained photo capture.

The process of syndication in TwiddleNet starts with a provisioning document. This provisioning document template is constructed such that the XML it produces adheres to the Atom standard; both in type integrity and in the observance of mandatory fields. Entry generation is commenced as a consequence of placing the file within a specific directory. This sharing action initiates application level processes that gather appropriate metadata and construct the baseline entry. Subsequent tag population and modification

continues as long as the file remains in a shared state, although the initial entry is correct to the Atom standard and can be transmitted without further modification.

Metadata customization is possible using the Atom standard through the use of XML namespace extensions. Atom parsers drop unknown tags by default, but can interpret other information contained in the entry. This facet of the Atom standard allows two key functions: 1) It enables customization of the TwiddleNet portal, since tags can be freely employed that have precise meaning within the TwiddleNet namespace, and 2) Clients not running the TwiddleNet application could still receive content by subscribing to posted feeds via an ordinary web browser. Content outreach is accordingly widened, while allowing the portal to react in a desired fashion to the posted metadata, according to administrator needs.

The TwiddleNet device application also maintains a complete Atom feed. The update mechanism for the feed is tied to the same processes that drive entry creation. As files are shared, unshared, deleted or modified, corresponding entries are handled appropriately; similarly the Atom feed is also modified to reflect shared folder contents. This document could then be passed to the portal as a complete feed detailing the device contents and viewable to any Atom capable browser.

G. ADD, DELETE, MODIFY

While not explicitly dealing with generation, maintaining metadata state awareness is necessary to insure information currency within TwiddleNet. This is achieved through the use of a special tag (hereafter referred to as the Action tag) attached to each generated metadata file. Simply put, the Action tag is updated according to actions taken either on the shared content or on the corresponding XML. When a file is first shared, it receives an Action value of add; modify and

delete values are applied as one would intuit. When metadata is posted to the TwiddleNet portal, the Action tag value determines how that metadata will be handled.

H. METADATA GENERATION PROCESS FLOW

This section details the tagging process flow of a photograph. The flow is depicted sequentially, from initial provisioning to completed metadata transmission.

1. Step One: Provisioning Document Setup

The provisioning document must first be set up to reflect mission requirements. This is accomplished using contextual and syntactic tag interfaces. Mandatory fields will be populated, and any automated tags the user wishes to appear in subsequent metadata documents will be selected.

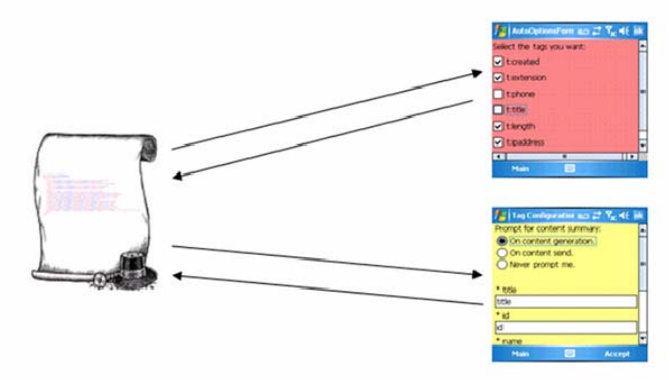

Figure 15. Provisioning document set up

2. Step Two: Content Generation

Once content is generated (depicted here as a photo of an automobile), a corresponding metadata file containing tags selected and populated in Step 1 is

created. That metadata file will hereafter be the file to which modifications are applied and that will eventually be sent to the TwiddleNet portal.

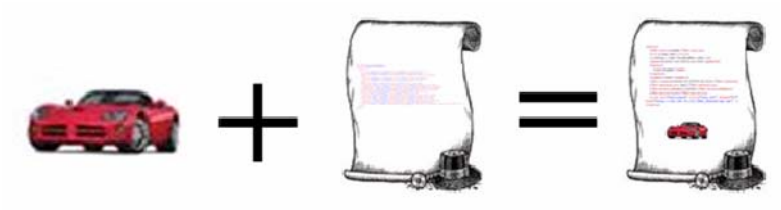

Figure 16. Metadata file creation based on shared photo

3. Step Three: OnGeneration Tagging and Metadata Evolution

Users may elect to append some contextual tags immediately after content capture (so called OnGeneration tags). If so, the content associated metadata file generated in Step 2 will be modified to reflect any additions entered into the OnGeneration user interface. Alternatively, they may elect to defer this process, or not to populate these tags at all. Metadata files may be freely modified at any point while the associated content is still in a shared mode.

Figure 17. OnGeneration tag population and ongoing modification

4. Step Four: Metadata Finalization and Transmission

Metadata is transmitted at varying times throughout application usage. Prior to transmission, some finalizing tasks must be accomplished. First, OnGeneration tags that had been deferred are offered for population. OnSend tags (customarily automated) are then added to the metadata. Finally, the document is scanned in its entirety, and unused tags and administrative attributes are stripped. The document has been slimmed to its bare essentials and is now ready to be transmitted.

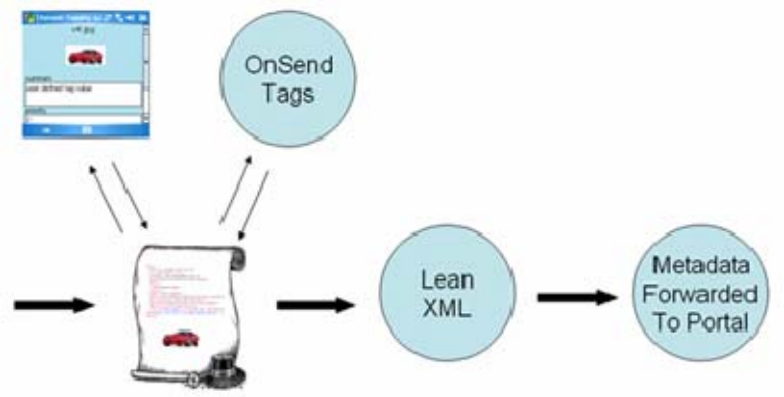

Figure 18. Metadata finalization and transmission

THIS PAGE INTENTIONALLY LEFT BLANK

V. DATA DISTRIBUTION

This chapter describes in detail how data is disseminated through the TwiddleNet network. Several cases will be discussed to include posting of original content, manual retrieval of content and automatic alerts.

A. DISTRIBUTION TECHNIQUES

TwiddleNet uses a hybrid peer-to-peer architecture using a combination of push and pull mechanisms to disseminate information among network users. The architecture is a hybrid P2P because the portal is a unique entity within the network. The portal performs the function of a central store for the network's shared metadata. Without the portal, no user would be able to locate information of interest. Nevertheless, it is a P2P architecture because once a user has located information of interest a direct P2P connection is made between the client and the personal server that contains the data.

1. Posting Content

The data dissemination process in TwiddleNet begins with a TwiddleNet user posting some content he wishes to share. Posting is accomplished by the user placing a document in the shared folder of the application and triggering the send function of the application. The send function performs an information push to the TwiddleNet portal. The function contains a series of criteria that must be met before the metadata document can be sent to the portal. The first check that takes place is to confirm that there is actually a document in the shared folder that needs to be sent. If confirmed, the application then checks to insure that the device has a connection. If the personal server has an existing connection, the signal strength is checked against a given threshold. If any of these checks fail, the application will not attempt to send the document. On the other hand, if all conditions are met, the application will attempt to post the document with the

portal. This posting is an HTTP POST request made to the portal. For a post to be accepted by the portal, the HTTP request must be properly formatted and the XML document must be a properly formatted *add* document with all required tag values filled. The final phase of posting is the portal returning an HTTP message that the document has been successfully added. The following figure shows a state diagram of the posting process.

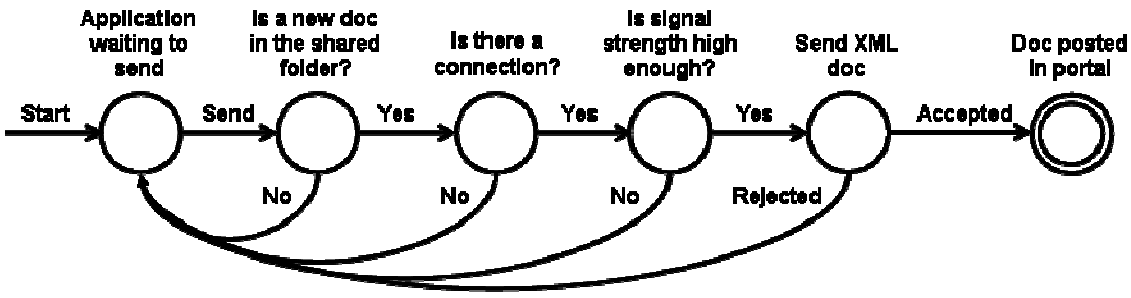

Figure 19. Posting process state diagram.

2. Viewing Shared Documents

There are several methods that enable the user to view shared information via TwiddleNet. The first is to browse to the portal and conduct a search for the content of interest. The second is to subscribe to information of interest and receive an alert when information becomes available.

A mobile client on TwiddleNet can view content once a document has been posted in the portal. The process of viewing shared documents involves an information pull from the portal to a mobile client (Step 1, Figure 21). The mobile client initiates the process by browsing to the portal's URL. The URL provides a search function where the user can search via keywords such as title, author, date, etc. The search function is similar to standard Web searches available today on the internet. The search results will provide the user with a list of documents and their corresponding tagged information as seen in the following figure.

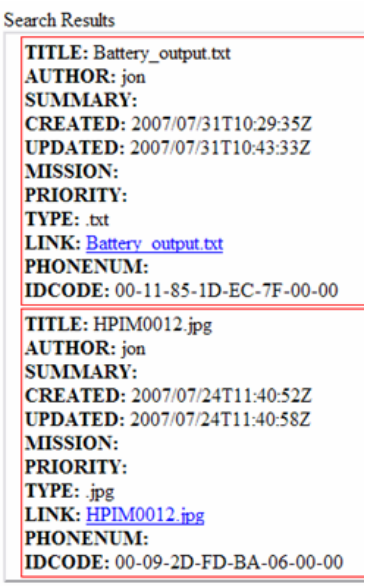

Figure 20. Sample result from a TwiddleNet search request.

From this point, to view a document the user simply clicks the link provided. This establishes the peer-to-peer connection from the client device to the server device (Step 2, Figure 21). Again HTTP is used to request the document. The client device initiates the HTTP session by sending a GET request to the client. The GET request contains the URL for the specified document. The server device then returns an HTTP POST message including the document.

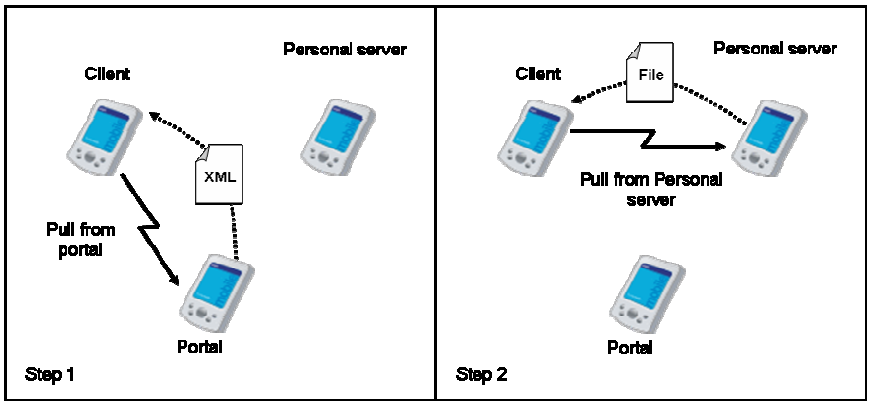

Figure 21. Pulling data from the portal and mobile server.

The second method to view data is to receive automatic alerts when new information is available. It uses a combination of push and pull techniques to disseminate the data through the network. This method is similar to publish-subscribe methods gaining popularity on the Web today. A subscriber places a standing request with the portal to be notified when specific documents of interest become available. The process begins when new content is generated and posted on the portal (Step 1, Figure 23). When the portal receives new content matching a subscriber's request, it will push an automated alert to the subscriber's device (Step 2, Figure 23). This alert contains the title, author and date/time group of the new document as seen in the below figure.

Figure 22. Automatic update from portal.

The user can choose whether to view the data or not. If he chooses to view the document, the client device will place an HTTP GET request to the URL in the alert (Step 3, Figure 23). The URL is the exact address for the newly generated document described in the alert. The process now precedes the same as before. The entire content generation with alert sequence is diagramed below.

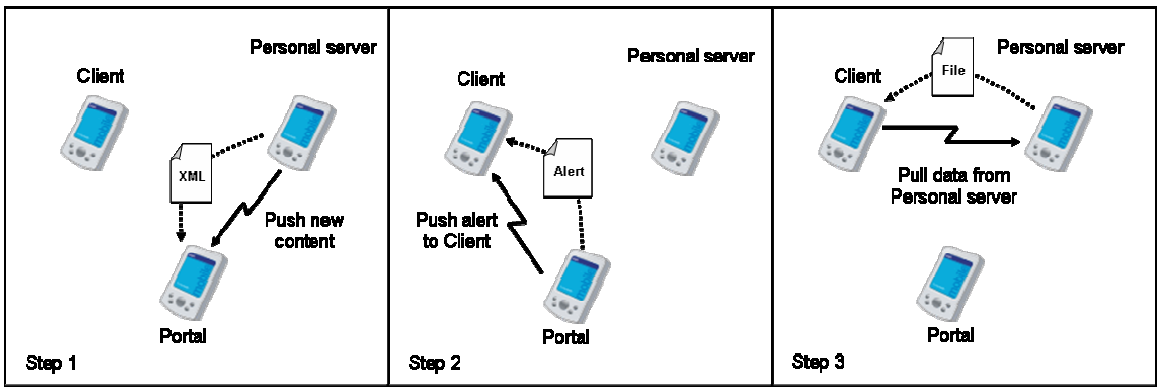

Figure 23. Alert push and data pull sequence

B. RESOURCE MANAGEMENT TECHNIQUES

Given the resource limitations of mobile devices, resource management and conservation are important issues in any mobile application. There are some features on devices that allow the user to adapt the settings to his/her liking such as adjustments of the backlight and WiFi power saving modes. Mobile device resource management must also take into consideration the resource management of the user. The most critical of these resources is time. Mobile device operators use mobile devices to help manage time and reduce their task load. Ultimately, the user has the best situational awareness as to how the device should be operating given the current circumstances. Therefore, he/she should be able to choose the settings to meet the current requirements. Yet, once set the device should remain hands off from a user's perspective. These techniques allow more user interaction and control over the device. The design of TwiddleNet incorporates several resource management features into the data dissemination schemes. These features aim at increasing the efficiency of the device and the user. This functionality takes into consideration the current context of the user and the device. For example, is the user busy performing other tasks? Does the user wish to share new content now or can it wait? Does

the device have a connection? These questions are addressed by allowing the user to manually set automatic functions that help control the device.

C. SENDING OCCASION OPTIONS

The user is provides with five different options for when they would like to send their updates. These options include sending on:

1. Timed interval.
2. Document generation.
3. Delayed document generation.
4. Sensing a connection.
5. Manual.

The *timed interval* option allows for documents to be sent on regular timed intervals such as every hour or every ten hours. The feature allows the user to define a sending interval between 1 minute and 999 hours. This feature will allow the user to collect documents during the time interval without having to manually send them. It also allows for the collection of many documents to be sent at one time vice being sent one at a time, thus conserving power and bandwidth.

The *on document generation* feature allows for automatic sending of documents as soon as they are either created or added to the shared folder. This feature allows the user to automatically update the portal with information as soon as it is created. Although this is not ideal for power conservation it is ideal in situations where information timeliness is an issue. The incorporation of the *on document generation* feature is what makes TwiddleNet a real-time application. The feature provides documents for viewing by others within a matter of seconds. This real-time ability is crucial in many military and disaster response applications.

The *on delayed generation* function is an attempt to gain the power saving benefits of sending numerous documents at once while still maintaining the real-time benefits of the application. In many situations where pictures are being taken, more than one picture is taken in the same relative time frame of the subject matter. The *on delayed generation* feature works by activating a timer set to five minutes when a document is generated. During the five minutes if another document is generated the timer is reset. This continues until the timer expires at which time all of the new documents are sent together. This feature is ideal when the user takes pictures in clusters. Say for instance a user saw a site or an object of interest and is going to take several pictures. The delayed sending feature allows all of the pictures to be collected and their documents to be sent after the picture taking was complete (i.e., the timer expired). This saves time, bandwidth and power by making one connection rather than numerous individual ones.

The sending on a *new connection* feature is designed for use in areas where Wi-Fi or cellular connections are sparse. This function frees the user from constantly monitoring a connection status to update information. The device passively monitors the connection status and when a connection is detected initiates the posting sequence.

The *manual* sending feature is always functional, offering the user the option to send at any given time. This enables the user to override any previously chosen sending algorithm if the user deems necessary.

The data sending features are designed to increase the overall efficiency of both the user and the device. Allowing the device to handle the menial tasks that require constant monitoring frees the user to concentrate on his immediate task. Furthermore, creating schemes that optimize device resources allows the device to perform its tasks for an increased amount of time.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SUMMARY AND CONCLUSIONS

TwiddleNet is a dynamic mobile file sharing application that enhances the functionality of today's highly capable mobile devices. Through dynamic document tagging and automated information dissemination, TwiddleNet increases today's mobile user's ability to instantly capture and share data. Further, functionality of TwiddleNet enables enhanced owner control over content accessibility. More importantly these functions are provided with an emphasis placed on conservation of device resources. It has been shown that numerous scenarios such as emergency response, military, law enforcement and social file sharing can all benefit from TwiddleNet. TwiddleNet's flexible and easily adaptable design allows for the seamless transition from its use as a tool for military forces on the battlefield to teenagers sharing pictures of the latest party.

The current development of TwiddleNet presents a robust mobile file sharing architecture that employs a dynamic XML tagging scheme. TwiddleNet optimizes metadata generation and minimizes the manual input burden to the user. TwiddleNet also employs a data dissemination scheme that is flexible enough to allow fine-grained user control while maximizing device resources at the same time. The current version of TwiddleNet serves as a solid foundation that positively demonstrates the functionality of the network architecture.

A. FUTURE WORK

The TwiddleNet system to date is probably best described as an early prototype. While a great many functions have been implemented, there still remains a substantial amount of work to do before scalability can be evaluated and full system testing can take place. The following sections discuss future design implementations and recommendations to improve the current iteration of TwiddleNet, or to further work already begun.

1. Portal Caching and Intelligent File Servicing

One goal of TwiddleNet is to implement distributed computing on edge devices to the largest extent possible. This has already been achieved to a large degree; file generation, metadata creation and file servicing functions are all resident on TwiddleNet devices. However, the mobile context presents some issues in regards to scalability. Although TwiddleNet devices are intelligent about metadata upload, they are not currently intelligent about file servicing. A TwiddleNet personal server will service all requests, current device state notwithstanding. While an implementation that would address this issue is clearly necessary, a great deal of pertinent code has already been implemented in this regard. The same evaluation algorithms applied to metadata upload can also be applied to the file servicing process. So while this is an important point, it may be relatively easily tackled.

Preventing file servicing based on personal server resource context does not solve the fundamental issue; requesting clients still have need of the file, even if intelligent servicing prevents its transmission. For example, suppose a particular TwiddleNet producer had generated a unique photograph of some occurrence. Suppose also that numerous service requests for that file had been received and honored by the owner personal server, until the personal server resource context prevented further servicing of the photo. Requests may continue to pour in, but the device is no longer capable of providing the file given its current resource state. Although device resources are now being intelligently conserved, client needs have not been fully met.

This situation could be resolved with content caching at the TwiddleNet portal. Via statistical methods, the need for that file could be assessed as so high as to warrant complete upload of the file to more robust storage. The file would then be available from a different location for which resource constraints are not an issue. Initial efforts in that direction have begun, although they are in their infancy. TwiddleNet producers can provide encoded photos as a file associated metadata tag for subsequent upload to the portal during a standard

update. However, this election is solely up to the user and is in no way automated. Such an implementation may greatly improve overall system performance and data reliability.

2. Syndication

Currently, TwiddleNet metadata is provided in a form that is compliant to the Atom standard. Each shared file represents an Atom formatted entry, or node, within the syndication feed, or tree. These entries are individually stored on each TwiddleNet producer device and are added to an Atom feed maintained on the same platform. This feed is designed to always reflect the current state of the shared directory.

Implementations to post device feeds on the TwiddleNet portal have not yet been developed. The portal is the only truly centralized entity in the TwiddleNet architecture and is therefore the most reasonable location for syndication feeds to be placed. Implementations could take several forms. The exact feed produced by the mobile device could be directly appended to the portal homepage, or customized feeds could be created out of entry level node collections extracted from the portal database. Either implementation would allow non-TwiddleNet users the ability to access personal server data via syndication subscriptions.

3. Subnets and Handoff

Currently TwiddleNet producers provide their IP address as the sole indication of their whereabouts. For architectures where devices are all collocated on the same subnet this is sufficient. On the other hand, for architectures where true mobility is desired this model must be changed. More thorough routing affiliated metadata must be provided from the device to ensure clients external to its subnet can be reached. Note that cellular datalink devices present special issues that effectively prevent disparate subnet file sharing at the

lowest level. So although subnet issues can be handled for Wi-Fi, cellular networks at this time remain out of reach.

Handoff is another issue that should be addressed in future iterations. Under the current implementation, TwiddleNet producers do not provide any sort of update resulting from handoff or change in IP address. Unless file associated metadata is specifically changed, or a UGC file is otherwise altered, the portal remains unaware of location changes, even if the last known address has become stagnant. Connection time outs resulting from stagnated pointers slow the network down substantially and preclude information exchange outright. This clearly goes against the TwiddleNet design principles of “real-time” content sharing, and is a key area of address for system improvement.

4. File Transfer

Although TwiddleNet can effectively tag any type of file, only files capable of being rendered in a browser are fully sharable. This is due to the implementation currently in place where files are requested and opened directly from such an application. This will require a change from the current model to one in which files are downloaded in a manner appropriate to their file type and to user desire. There is nothing wrong with rendering images for viewing in a browser, but if the user wishes to zoom in on a certain part of the image, then the image must be otherwise accessible. The current design is very limiting in this regard, and a change to the mode in which files are downloaded will go a great distance in improving system flexibility.

5. System Log In

The system does not provide any means for users to log in and remain known to the portal. This prevents discrete management of content alerts to specific consumers. Currently the system simply issues alerts to all active members about all posted content. Since there is no way to identify members or to recall information about them, this is the only means available to perform such

an information push. This implementation is wasteful. By updating every user, the portal wastes time and bandwidth attempting to connect to offline users. A system log in will allow user identification to be associated with content descriptors, allowing a more streamlined implementation of the alert system.

6. Security

The only security mechanism employed by the TwiddleNet architecture to date is that provided by virtue of the carrier network. In the case of WiFi for instance, security is most often Wired Equivalent Privacy (WEP) and generally nothing more. Given the usage model in which this system would be used, robust security is a must. Privacy, Integrity, authenticity and provision of service are only a few of the areas that need to be addressed. Furthermore, these issues must be addressed from the perspective of the personal server and the portal. Personal server protection is of the utmost importance since users allow unknown visitors direct access to their device. It is crucial to ensure that access is strictly limited to the TwiddleNet shared directory, thereby precluding malicious intrusion to other parts of the device. Portal protection is also a critical concern, since it is a potential single point of failure in the architecture.

7. Single Point of Failure

Since TwiddleNet is a hybrid peer-to-peer architecture it contains a unique network entity, the portal. Currently, the portal is a single data base that contains all of the shared metadata provided by the personal servers. Therefore the portal is a single point of failure and is a critical target for malicious users and a vulnerability of the network in the case of device failure or malfunction. There has been significant work in the field of distributed databases and mobility. The approach of spreading the database content among numerous separate network entities would be a large step forward to protecting the portal through redundancy.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns:t="http://www.TwiddleNet.com">
  <action action="">
    <entry>
      <title when ="predefined" how="userdefined" authority="mandatory" dataType ="0"></title>
      <id when ="predefined" how="userdefined" authority="mandatory" dataType ="0"></id>
      <updated when ="onSend" how="automatic" selected="true" authority="mandatory" dataType ="2"></updated>
      <author>
        <name when ="predefined" how="userdefined" authority="mandatory" dataType ="0"></name>
        <email when ="predefined" how="userdefined" authority="optional" dataType ="1"></email>
        <url when ="predefined" how="userdefined" authority="optional" dataType ="4"></url>
      </author>
      <summary when ="onGen" how="userdefined" authority="optional" dataType ="0"></summary>
      <t:created when ="onGen" how="automatic" selected="true" authority="mandatory" dataType ="2"></t:created>
      <t:fileUpdated when ="onGen" how="automatic" selected="true" authority="optional" dataType ="2"></t:fileUpdated>
      <t:extension when ="onGen" how="automatic" selected="true" authority="optional" dataType ="6"></t:extension>
      <t:missionNumber when ="predefined" how="userdefined" authority="optional" dataType ="0"> </t:missionNumber>
      <t:priority when ="onGen" how="userdefined" authority="optional" dataType ="6"></t:priority>
      <t:kw when ="onGen" how="userdefined" authority="optional" dataType ="0"></t:kw>
      <t:idcode when ="onSend" how="automatic" selected="true" authority="mandatory" dataType ="6"></t:idcode>
      <t:phone when ="onSend" how="automatic" selected="true" authority="optional" dataType ="7"></t:phone>
      <t:title when ="onGen" how="automatic" selected="true" authority="mandatory" dataType ="0"></t:title>
      <t:length when ="onGen" how="automatic" selected="true" authority="optional" dataType ="0"></t:length>
      <t:ipaddress when ="onSend" how="automatic" selected="true" authority="optional" dataType ="0"></t:ipaddress>
      <t:image when ="onGen" how="automatic" selected="true" authority="optional" dataType ="0"></t:image>
      <link when ="onSend" how="automatic" selected="true" authority="mandatory"  dataType ="5" rel="enclosure" title=""
length="" href=""></link>
    </entry>
  </action>
</feed>
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] T. Pering, Y. Agarwal, R. Gupta and R. Want, "CoolSpots: reducing the power consumption of wireless mobile devices with multiple radio interfaces," in MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services, pp. 220-232, 2006.

[2] Semagix White Paper, "A Semantic Metadata Approach to Enterprise Information Integration."

[3] H. Halpin, V. Robu and H. Shepherd, "The Complex Dynamics of Collaborative Tagging," in WWW 2007, pp. 211-211-220, 2007.

[4] D. Johnson, RSS and Atom in action, 209 Bruce Park Ave., Greenwich, CT 06830: Manning Publications Co., 2006, pp. 368.

[5] M. Nottingham and R. Sayre, "RFC 4287; The Atom Syndication Format," pp. 1-43, 2005.

[6] A. Sorvari, J. Jalkanen, R. Jokela, A. Black, K. Koli, M. Moberg and T. Keinonen, "Usability issues in utilizing context metadata in content management of mobile devices," in NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction, pp. 357-363, 2004.

[7] M. Ames and M. Naaman, "Why we tag: motivations for annotation in mobile and online media," in CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 971-980, 2007.

[8] R. Sarvas, E. Herrarte, A. Wilhelm and M. Davis, "Metadata creation system for mobile images," in MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services, pp. 36-48, 2004.

[9] G. Muhl, A. Ulbrich and K. Herrman, "Disseminating information to mobile clients using publish-subscribe," Internet Computing, IEEE, vol. 8, pp. 46-53, 2004.

[10] A.P. Afonso and M.J. Silva, "Dynamic Information Dissemination to Mobile Users," Mobile Networks and Applications, vol. 9, pp. 529, Oct. 2004.

[11] K. Ragab, N.Y. Horikoshi, H. Kuriyama and K. Mori, "Autonomous decentralized community communication for information dissemination," Internet Computing, IEEE, vol. 8, pp. 29-36, 2004.

[12] A. Oram, "Peer-to-peer: harnessing the benefits of a disruptive technology," pp. 432, 2001.

[13] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications," in Procedings of the First International Conference on Peer-to-Peer Computing, 2001.

[14] M. Mallick, "Mobile and wireless design essentials," pp. 454, 2003.

[15] O. Volgin, W. Hung, C. Vakili, J. Flinn and K.G. Shin, "Context-aware metadata creation in a heterogeneous mobile environment," in NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video, pp. 75-80, 2005.

[16] P. Vartiainen, "Using Metadata and Context Information in Sharing Personal Content of Mobile Users," 27 Feb 2003. 2003.

[17] J. Wikman and F. Dosa, "Providing HTTP Access to Web Servers Rinning," Nokia., Tech. Rep. NRC-TR-2006-005, 2006.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California